MethodSCRIPT Examples - iOS

PalmSens
Compact Electrochemical Interfaces

Last revision: October 10, 2021

www.palmsens.com

1.1 Introduction

The example in the 'MethodSCRIPTExamples_iOS' folder demonstrates basic communication with the EmStat Pico from an iOS device using Bluetooth Low Energy (BLE). The example shows how to connect to an EmStat Pico or other MethodSCRIPT supporting device, send a MethodSCRIPT and read and parse the measurement data packages using Apple's programming language Swift. The UI was built with SwiftUI.

1.2 Prerequisites

1.2.1 For EmStat Pico Development Board

To connect via Bluetooth LE, make sure that the UART switch block SW4 on the EmStat Pico development board has the switches for Bluetooth 5 and 6 turned on. Also, the UART Switch block SW3 should have the switches for PICO Bluetooth 5 and 6 turned on.

1.2.2 For all PalmSens OEM devices

The Dual Mode smartBASIC script needs to be running on the Laird BT900 Bluetooth module in the PalmSens Device. With the Dual Mode script the device presents itself as a classic Serial Port Profile (SPP) Bluetooth device as well as a Bluetooth LE (BLE) device. iOS devices can only work with the BLE profile. In case your PalmSens device does not show as a Bluetooth device on your iOS device, please follow this tutorial to re-program the Bluetooth settings on your PalmSens device:
www.palmsens.com/knowledgebase-article/change-bluetooth-settings

1.3 The Example Project

The example project was created with Xcode version 11.3.1. Open project 'MethodSCRIPTExample_iOS.xcodeproj' residing in folder 'MethodSCRIPTExample_iOS'.

The first time the project is opened, you must adjust the Team, see the 'Signing & Capabilities' tab, to your team (it is now attached to a PalmSens team member).

The most important project files are:

File 'BLEConnection.swift'
This file contains all the necessary functions to connect, read, write to the EmStat Pico over Bluetooth LE. The file also contains functionality to parse the data from the device. Most important functions:

Name	Description
startScanningForDevices	Starts scanning for devices. The list of devices scanned for is limited by filtering on the service UUID. See constant 'vspServiceUUID'.
connectToDevice	After selecting a device from the app, this method is called. This method stops scanning for devices and connects to the selected device.
Disconnect	Disconnects from a device.
sendCommandVersion	Sends the command 't\n' to the device. The response should be the device version. For example: 'testpico#Aug 2 2019 11:06:07'.
sendMethodScript	Sends a methodscript, which starts the measurement defined in the script. The device should respond with measurement packages. The functions 'sendMessage', 'sendBytes' and

	'sendData' are all helper methods to support this function. Note data is written in chunks of 20 bytes. Function 'sendData' handles this.
processReceivedPackage	Handles the data read from the device. The data is buffered and parsed in a First In First Out (FIFO) way.
parsePackageLine	Analyses a measurement package. The measurement package contains the voltage, current and metadata. The functions parseDataValue, parseMetadata, parseCurrentRange and parseNoise are all helper methods to parse the different parts of the measurement package.

The functions from the comment 'Mark' section 'Central Manager Methods' are functions taken from examples found on the internet.

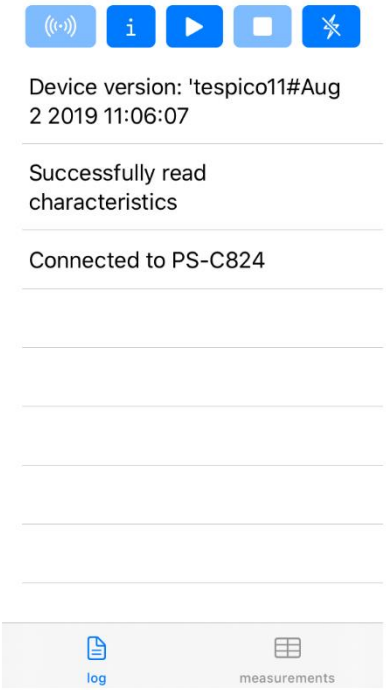
File 'Info.plist'
The key 'NSBluetoothAlwaysUsageDescription' was added to let the user enable Bluetooth when opening the app for the first time.

File 'ContentView.swift'
The app is using SwiftUI for creating the UI. SwiftUI is at the moment of writing a new technology from Apple, and some UI controls are rather limited. For example, an out of the box 'Excel-like' grid is missing for showing the measurements. This grid is now mimicked using a List. You can mix SwiftUI with UIKit if you need a more advanced UI experience.

The view contains two views:

1. The main 'ContentView' view which contains on top 5 buttons: the first button is for scanning for devices, the second button is for getting the device information, the third button for starting a measurement, the fourth button stops a running measurement, and with the last button you can disconnect from a connected device.
2. The child 'ScanForDevicesView' view, which shows the list of found devices, from this view you can connect to a device.

A screenshot from the app:



Below there are two tabs; the first tab shows the log and the second tab the result of a measurement. When you test using Xcode, there is more detailed logging available in the output window.

1.3.1 Sending a MethodSCRIPT

A method script is sent when pressing the play button. The script is defined in a constant (a Swift 'let' variable) named 'script', see 'ContentView.swift' where a button action starts:

```
self.bleConnection.sendMethodScript(methodScript: script)
```

In the script a Chronoamperometry measurement is started for 5 seconds. Around 200 points per second can be measurement without any noticeable errors (testresults on an iPhone SE and on an iPhone 7).

1.3.2 Parsing the measurement packages

The measurement data packages can be parsed further to obtain the actual data values. Here's a set of data packages received from a Linear Sweep Voltammetry (LSV) measurement on a dummy cell with 10kOhm resistance.

```
e\n
M0000\n
Pda7F85F3Fu;ba48D503Dp,10,288\n
Pda7F9234Bu;ba4E2C324p,10,288\n
Pda806EC24u;baAE16C6Dp,10,288\n
Pda807B031u;baB360495p,10,288\n
*\n
\n
```

While parsing a measurement package, various identifiers are used to identify the type of package. For example, In the above sample,

1. ‘e’ is the confirmation of the “execute MethodSCRIPT” command.
2. ‘M’ marks the beginning of a measurement loop.
3. ‘P’ marks the beginning of a measurement data package.
4. “*\n” marks the end of a measurement loop.
5. “\n” marks the end of the MethodSCRIPT.

The data values to be received from a measurement can be sent through ‘pck’ commands in the MethodSCRIPT. Most techniques return the data values Potential (set cell potential in V) and Current (measured current in A). These can be sent with the MethodSCRIPT.

In the case of Electrochemical Impedance Spectroscopy (EIS) measurements, the following *variable types* can be sent with the MethodSCRIPT and received as measurement data values.

- Frequency (set frequency in Hz)
- Real part of complex Impedance (measured impedance Ohm)
- Imaginary part of complex Impedance (measured impedance in Ohm)

The following metadata values can also be obtained from the data packages if present.

- CurrentStatus (OK, underload, overload, overload warning)
- CurrentRange (the current range in use)
- Noise (Noise)

1.3.2.1 Parsing the measurement data values

Each measurement data package begins with the header ‘P’ and is terminated by a ‘\n’. The measurement data package can be split into data value packages based on the delimiter ‘;’. Each of these data value packages can then be parsed separately to get the actual data values.

The type of data in a data package is identified by its variable type:

- The potential readings are identified by the string *da*
- The current readings are identified by the string *ba*
- The frequency readings are identified by the string *dc*
- The real impedance readings are identified by the string *cc*
- The imaginary impedance readings are identified by the string *cd*

For example, in the sample package seen above, the *variable types* are
da7F85F3Fu - da Potential reading and
ba48D503Dp,10,288 – ba current reading.

The following eight characters hold the data value. The data value for the current reading (8 characters) from the above sample package is *48D503Dp*.

The SI unit prefix from the package can be obtained from the parameter value at position 8

In the above sample package, the unit prefix for current data is ‘p’ which is 1e-12 A.

After obtaining the variable type and the data values from the package, the metadata values can be parsed, if present.

Please note frequency, real impedance and imaginary impedance readings are not implemented in this example.

1.3.2.2 Parsing the metadata values

The metadata values are separated based on the delimiter ',' and each of the values is further parsed to get the actual value.

The first character of each metadata value metaData[0] identifies the type of metadata.

'1' – status
'2' – Current range index
'4' - Noise

The metadata status is a one-character hexadecimal bitmask.

For example, in the above sample, the available metadata values for current data are,
10,288. The first metadata value is 10.

1 – metadata status – 0 indicates OK.

The metadata type current range is two characters long hexadecimal value. If the first bit is high (0x80), it indicates a high-speed mode current range. The hexadecimal value can be converted to int to get the current range.

For example, in the above sample, the second metadata available is 288.

2 – indicates the type – current range
88 – indicates the hexadecimal value for current range index – 1mA. The first bit 8 implies that it is a high-speed mode current range.

Please note noise readings are not further processed in this example.

Sample output:

Here's a sample measurement data package from an LSV measurement on a dummy cell with 10 kOhm resistance and its corresponding output.

Pda7F85F3Fu;ba4BA99F0p,10,288

Output: E (V) = -4.999E-01
i (A) = -4.999E-01
Status = OK
CR = 1mA (High speed)